# Remote Procedure Call

***Software Technology Roadmap***

## Status

Advanced

## Note

*We recommend* Middleware *as prerequisite reading for this technology description.*

## Purpose and Origin

Remote Procedure Call (RPC) is a client/server infrastructure that increases the *interoperability*, *portability*, and *flexibility* of an application by allowing the application to be distributed over multiple heterogeneous platforms. It reduces the *complexity* of developing applications that span multiple operating systems and network protocols by insulating the application developer from the details of the various operating system and network interfaces--function calls are the programmer's interface when using RPC [Rao 1995].

The concept of RPC has been discussed in literature as far back as 1976, with full-scale implementations appearing in the late 1970s and early 1980s [Birrell 84].

## Technical Detail

In order to access the remote server portion of an application, special function calls, RPCs, are embedded within the client portion of the client/server application program. Because they are embedded, RPCs do not stand alone as a discreet middleware layer. When the client program is compiled, the compiler creates a local stub for the client portion and another stub for the server portion of the application. These stubs are invoked when the application requires a remote function and typically support synchronous calls between clients and servers. These relationships are shown in Figure 32 [Steinke 95].

By using RPC, the complexity involved in the development of distributed processing is reduced by keeping the semantics of a remote call the same whether or not the client and server are collocated on the same system. However, RPC increases the involvement of an application developer with the complexity of the master-slave nature of the client/server mechanism.

RPC increases the flexibility of an architecture by allowing a

client component of an application to employ a function call to access a server on a remote system. RPC allows the remote component to be accessed without knowledge of the network address or any other lower-level information. Most RPCs use a synchronous, request-reply (sometimes referred to as "call/wait") protocol which involves blocking of the client until the server fulfills its request. Asynchronous ("call/nowait") implementations are available but are currently the exception.

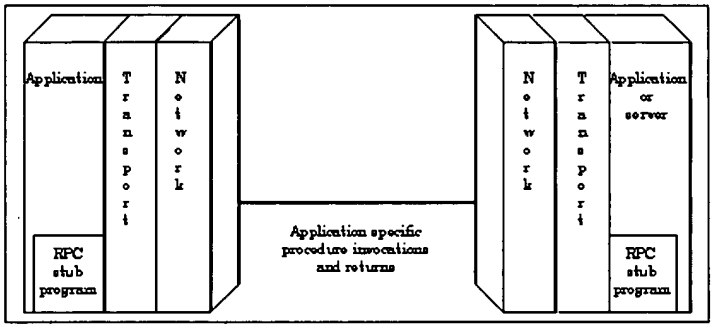


**Figure 32: Remote Procedure Calls**

RPC is typically implemented in one of two ways:

1. within a broader, more encompassing propriety product
2. by a programmer using a proprietary tool to create client/server RPC stubs

## Usage Considerations

RPC is appropriate for client/server applications in which the client can issue a request and wait for the server's response before continuing its own processing. Because most RPC implementations do not support peer-to-peer, or asynchronous, client/server interaction, RPC is not well-suited for applications involving distributed objects or object-oriented programming (see Object-Oriented Programming Languages).

Asynchronous and synchronous mechanisms each have strengths and weaknesses that should be considered when designing any specific application. In contrast to asynchronous mechanisms employed by Message-Oriented Middleware, the use of a synchronous request-reply mechanism in RPC requires that the client and server are always available and functioning (i.e., the client or server is not blocked). In order to allow a client/server application to recover from a blocked condition, an implementation of a RPC is required to provide mechanisms such as error messages, request timers, retransmissions, or redirection to an alternate server. The complexity of the application using a RPC is dependent on the sophistication of the specific RPC implementation (i.e., the more sophisticated the recovery mechanisms supported by RPC, the less complex the application utilizing the RPC is required to be). RPCs that implement asynchronous mechanisms are very few and are difficult (complex) to implement [Rao 1995].

When utilizing RPC over a distributed network, the performance

(or load) of the network should be considered. One of the strengths of RPC is that the synchronous, blocking mechanism of RPC guards against overloading a network, unlike the asynchronous mechanism of Message-Oriented Middleware (MOM). However, when recovery mechanisms, such as retransmissions, are employed by an RPC application, the resulting load on a network may increase, making the application inappropriate for a congested network. Also, because RPC uses static routing tables established at compile-time, the ability to perform load balancing across a network is difficult and should be considered when designing an RPC-based application.

## Maturity

Tools are available for a programmer to use in developing RPC applications over a wide variety of platforms, including Windows (3.1, NT, 95), Macintosh, 26 variants of UNIX, OS/2, NetWare, and VMS [Steinke 1995]. RPC infrastructures are implemented within the Distributed Computing Environment (DCE) , and within Open Network Computing (ONC), developed by Sunsoft, Inc. These two RPC implementations dominate the current Middleware market [Rao 1995].

## Costs and Limitations

RPC implementations are nominally incompatible with other RPC implementations, although some are compatible. Using a single implementation of a RPC in a system will most likely result in a dependence on the RPC vendor for maintenance support and future enhancements. This could have a highly negative impact on a system's flexibility, maintainability, portability, and interoperability.

Because there is no single standard for implementing an RPC, different features may be offered by individual RPC implementations. Features that may affect the design and cost of a RPC-based application include the following:

- support of synchronous and/or asynchronous processing
- support of different networking protocols
- support for different file systems
- whether the RPC mechanism can be obtained individually, or only bundled with a server operating system

Because of the complexity of the synchronous mechanism of RPC and the proprietary and unique nature of RPC implementations, training is essential even for the experienced programmer.

## Alternatives

Other middleware technologies that allow the distribution of processing across multiple processors and platforms are

- Object Request Brokers (ORB)

- Distributed Computing Environment (DCE)
- Message-Oriented Middleware (MOM)
- COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities)
- Transaction Processing Monitor Technology
- Three Tier Software Architectures

## Complementary Technologies

RPC can be effectively combined with Message-Oriented Middleware (MOM)- MOM can be used for asynchronous processing.

## Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

| Name of technology | Remote Procedure Call |
|---|---|
| Application category | Client/Server (AP.2.1.2.1)<br>Client/Server Communication (AP.2.2.1) |
| Quality measures category | Maintainability (QM.3.1)<br>Interoperability (QM.4.1)<br>Portability (QM.4.2)<br>Complexity (QM.3.2.1) |
| Computing reviews category | Distributed Systems (C.2.4) |

## References and Information Sources


[Birrell 84] Birrell, A.D. & Nelson, B.J. "Implementing Remote Procedure Calls." *ACM Transactions on Computer Systems 2,* 1 (February 1984): 39-59.

**[Rao 95]** Rao, B.R. "Making the Most of Middleware." *Data Communications International 24*, 12 (September 1995): 89-96.

**[Steinke 95]** Steinke, Steve. "Middleware Meets the Network." *LAN: The Network Solutions Magazine 10*, 13 (December 1995): 56.

[Thekkath 93] Thekkath, C.A. & Levy, H.M. "Limits to Low-Latency Communication on High-Speed Networks." *ACM Transactions on Computer Systems 11*, 2 (May 1993): 179-203.


## Current Author/Maintainer

Cory Vondrak, TRW, Redondo Beach, CA

## Modifications

25 June 97: modified/updated OLE/COM reference to COM/DCOM
10 Jan 97 (original)

---

## Glossary Term

*Interoperability*

the ability of two or more systems or components to exchange information and to use the information that has been exchanged [IEEE 90].

## Glossary Term

*Portability*

the ease with which a system or component can be transferred from one hardware or software environment to another [IEEE 90].

## Glossary Term

*Flexibility*

the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed [IEEE 90].

## Glossary Term

*Complexity*

1. (Apparent) the degree to which a system or component has a design or implementation that is difficult to understand and verify [IEEE 90].
2. (Inherent) the degree of complication of a system or system component, determined by such factors as the number and intricacy of interfaces, the number and intricacy of conditional branches, the degree of nesting, and the types of data structures [Evans 87].

## References and Information Sources

**[Rao 95]** Rao, B.R. "Making the Most of Middleware." *Data Communications International 24*, 12 (September 1995): 89-96.

## References and Information Sources

[Birrell 84] Birrell, A.D. & Nelson, B.J. "Implementing Remote Procedure Calls." *ACM Transactions on Computer Systems 2,* 1 (February 1984): 39-59.

## References and Information Sources

**[Steinke 95]** Steinke, Steve. "Middleware Meets the Network." *LAN: The Network Solutions Magazine 10*, 13 (December 1995): 56.

# Distributed Computing Environment

***Software Technology Roadmap***

## Status

Advanced

## Note

*We recommend* Middleware *as prerequisite reading for this technology description.*

## Purpose and Origin

Developed and maintained by the Open Systems Foundation (OSF), the Distributed Computing Environment (DCE) is an integrated distributed environment which incorporates technology from industry. The DCE is a set of integrated system services that provide an *interoperable* and *flexible* distributed environment with the primary goal of solving interoperability problems in heterogeneou networked environments.

OSF provides a reference implementation (source code) on which all DCE products are based [OSF 96a].The DCE is *portable* and flexible- the reference implementation is independent of both networks and operating systems and provides an architecture in which new technologies can be included, thus allowing for future enhancements. The intent of the DCE is that the reference implementation will include mature, proven technology that can be used in parts- individual services- or as a complete integrated infrastructure.

The DCE infrastructure supports the construction and integration of client/server applications whil attempting to hide the inherent *complexity* of the distributed processing from the user [Schill 93]. The OSF DCE is intended to form a comprehensive software platform on which distributed applications can be built, executed, and maintained.

## Technical Detail

The DCE architecture is shown in Figure 10 [Schill 93].

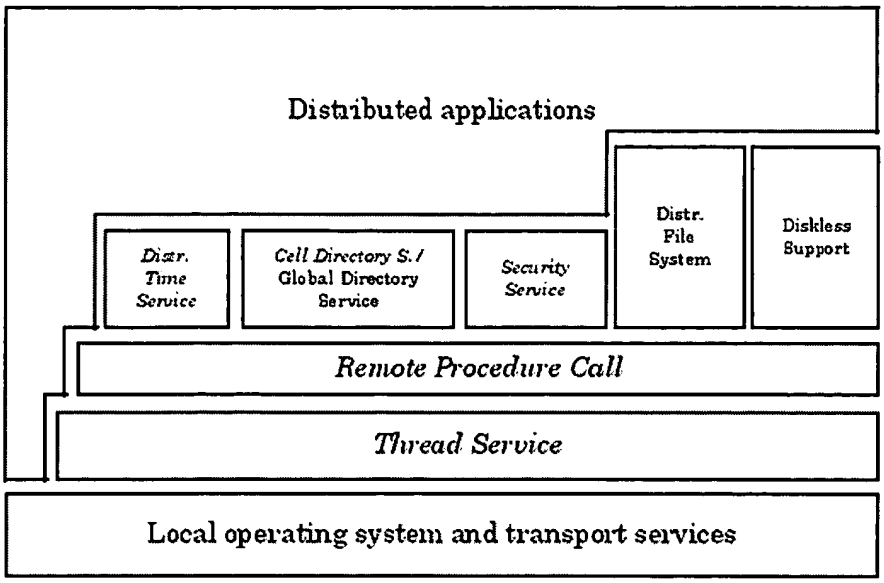


**Figure 10: Distributed Computing Environment Architecture**

DCE services are organized into two categories:

1. *Fundamental distributed services* provide tools for software developers to create the end-us services needed for distributed computing. They include
   - Remote Procedure Call, which provides portability, network *independence*, and *secure* distributed applications.
   - Directory services, which provide full X.500 support and a single naming model to allo programmers and maintainers to identify and access distributed resources more easily
   - Time service, which provides a mechanism to monitor and track clocks in a distributed environment and accurate time stamps to reduce the load on system administrator.
   - Security service, which provides the network with authentication, authorization, and us account management services to maintain the *integrity*, privacy, and authenticity of the distributed system.
   - Thread service, which provides a simple, portable, programming model for building concurrent applications.
2. *Data-sharing services* provide end users with capabilities built upon the fundamental distributed services. These services require no programming on the part of the end user anc facilitate better use of information. They include
   - Distributed file system, which interoperates with the network file system to provide a high-performance, *scalable*, and secure file access system.
   - Diskless support, which allows low-cost workstations to use disks on servers, possibly reducing the need/cost for local disks, and provides performance enhancements to reduce network overhead.

The DCE supports International Open Systems Interconnect (OSI) standards, which are critical to global interconnectivity. It also implements ISO standards such as CCITT X.500, Remote Operations Service Element (ROSE), Association Control Service Element (ACSE), and the ISO session and presentation services. The DCE also supports Internet standards such as the TCP/IF transport and network protocols, as well as the Domain Name System and Network Time Protoco provided by the Internet.

## Usage Considerations

The DCE can be used by system vendors, software developers, and end users. It can be used on any network hardware and transport software, including TCP/IP, OSI, and X.25. The DCE is writt in standard C and uses standard operating system service interfaces like POSIX and X/Open guidelines. This makes the DCE portable to a wide variety of platforms. DCE allows for the

extension of a network to large numbers of nodes, providing an environment capable of supportin networks of numerous low-end computers (i.e., PCs and Macintosh machines), which is importan downsizing and distributing of processing is desired. Because DCE is provided in source form, it can be tailored for specific applications if desired [OSF 96a].

DCE works internally with the client/server model and is well-suited for development of applicatior that are structured according to this model. Most DCE services are especially optimized for a structuring of distributed computing systems into a "cell" (a set of nodes/platforms) that is manage together by one authority.

For DCE, intra-cell communication is optimized and relatively secure and transparent. Inter-cell communication, however, requires more specialized processing and more complexity than its intra-cell counterpart, and requires a greater degree of programming expertise.

When using the thread services provided by DCE, the application programmer must be aware of thread synchronization and shared data across threads. While different threads are mutually asynchronous up to a static number defined at initialization, an individual thread is synchronous. The complexity of thread programming should be considered if these services are to be used.

DCE is being used or is planned for use on a wide variety of applications, including the following:

- *The Common Operating Environment.* DCE has been approved by DISA (Defense Information Systems Agency) as the distributed computing technology for the Common Operating Environment (COE) (see Defense Information Infrastructure Common Operating Environment).
- *The Advanced Photon Source (APS) system.* This is a synchrotron radiation facility under construction at Argonne National Laboratory.
- *The Alaska Synthetic Aperture Radar Facility (ASF).* This is the ground station for a set of earth-observing radar spacecraft, and is one of the first NASA projects to use DCE in an operational system.
- *The Deep Space Network's Communications Complexes Monitor and Control Subsystem.* This project is deploying DCE for subsystem internal communications, with the expectation that DCE will eventually form the infrastructure of the entire information system.
- *The Multimission Ground Data System Prototype.* This project evaluated the applicability of DCE technology to ground data systems for support of JPL flight projects (Voyager, Cassini, Mars Global Surveyor, Mars Pathfinder).
- *Earth Observing Systems Data Information System.* This NASA system is one of the largest information systems ever implemented. The system is comprised of legacy systems and dat computers of many varieties, networks, and satellites in space.
- *Command and control prototypes.* MITRE has prototyped command and control (C2) applications using DCE technology. These applications provide critical data such as unit strength, supplies, and equipment, and allow staff officers to view maps of areas of operatio [OSF 96b].

## Maturity

In early 1992, the OSF released the source code for DCE 1.0. Approximately 12 vendors had ported this version to their systems and had DCE 1.0 products available by June 1993. Many of these original products were "developer's kits" that were not robust, did not contain the entire set DCE features (all lacked distributed file services), and were suited mostly for UNIX platforms [Chappell 93].

The DCE continues to evolve, but many large organizations have committed to basing their next generation systems on the DCE- over 14 major vendors provided DCE implementations by late 1994, when DCE 1.1 was released.

DCE 1.2.1, released in March 1996, provided the following new features:

- Interface definition language (IDL) support for C++ to include features such as inheritance a

object references in support of object-oriented applications. This feature supports adoption c any object model or class hierarchy, thus providing developers with additional flexibility.
- Features to provide for coexistence with other application environments.
- Improvements over DCE 1.1 including enhancements to achieve greater reliability and bette performance [OSF 96a].

Two other approaches to supporting objects are being considered besides the approach describe for DCE 1.2:

1. Installing a CORBA-based product over DCE to provide additional support for distributed object technologies and a wide range of standardized service interfaces.
2. Integrating Network COM/DCOM (see Component Object Model (COM), DCOM, and Relate Capabilities) into the DCE infrastructure.

## Costs and Limitations

DCE was not built to be completely object-oriented. The standard interfaces used by the DCE, as well as all the source code itself, are defined only in the C programming language. For object-oriented applications (i.e., applications being developed using an object-oriented language (see Object-Oriented Programming Languages) such as C++ or Ada 95, it may be more complex, less productive (thus more expensive), and less maintainable to use a non-object-oriented set of services like the DCE [Chappell 96].

Object-oriented extensions of the DCE have been developed by industry, but an agreed to vendor-neutral standard was still being worked in 1996.

## Dependencies

Dependencies include Remote Procedure Call (RPC).

## Alternatives

Alternatives include CORBA (see Common Object Request Broker Architecture), COM/DCOM (se Component Object Model (COM), DCOM, and Related Capabilities), and message-oriented middleware (see Message-Oriented Middleware).

## Complementary Technologies

DCE, in-part, has been used in building CORBA-compliant (see Common Object Request Broker Architecture) products as early as 1995. OSF is considering support for objects using COM/DCOI (see Component Object Model (COM), DCOM, and Related Capabilities).

## Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

| Name of technology | Distributed Computing Environment |
|---|---|
| Application category | Distributed Computing (AP.2.1.2) |
| Quality measures category | Interoperability (QM.4.1)<br>Portability (QM.4.2)<br>Scalability (QM.4.3)<br>Security (QM.2.1.5)<br>Maintainability (QM.3.1)<br>Complexity (QM.3.2.1)<br>Throughput (QM.2.2.3) |
| Computing reviews category | Distributed Systems (C.2.4) |

## References and Information Sources


[Brando 96] Brando, T. "Comparing CORBA & DCE." *Object Magazine 6*, 1 (March 1996): 52-7.

[Chappell 93] Chappell, David. "OSF's DCE and DME: Here Today?" *Business Communications Review 23,* 7 (July 1993): 44-8.

**[Chappell 96]** Chappell, David. *DCE and Objects* [online]. Available WWW <URL: http://www.opengroup.org/dce/info/dce_objects.htm> (1996).

**[OSF 96a]** Open Software Foundation. *The OSF Distributed Computing Environment* [online]. Available WWW <URL: http://www.osf.org/dce/> (1996).

[OSF 96b] Open Software Foundation. *The OSF Distributed Computing Environment: End-User Profiles* [online]. Available WWW URL: < http://www.osf.org/comm/lit/dce-eup/> (1996).

[Product 96] *DCE Product Survey Report* [online]. Available WWW <URL: http://nsdir.cards.com/Libraries/HTML/PDLC/DCE_prod_surv_rpt.html (1996).

[Schill 93] Schill, Alexander. "DCE-The OSF Distributed Computing Environment Client/Server Model and Beyond," 283. *International DCE Workshop.* Karlsruhe, Germany, October 7-8, 1993. Berlin, Germany: Springer-Verlag, 1993.


## Current Author/Maintainer

Cory Vondrak, TRW, Redondo Beach, CA

## Modifications

25 June 97: modified/updated OLE/COM reference to COM/DCOM
10 Jan 97 (original)

---

URL: http://www.sei.cmu.edu/str/descriptions/dce_body.html
Last Modified: 3 September 2003

# Middleware

***Software Technology Roadmap***

## Status

Advanced

## Purpose and Origin

Middleware is connectivity software that consists of a set of enabling services that allow multiple processes running on one or more machines to interact across a network. Middleware is essential to migrating mainframe applications to client/server applications and to providing for communication across heterogeneous platforms. This technology has evolved during the 1990s to provide for *interoperability* in support of the move to client/server architectures (see Client/Server Software Architectures). The most widely-publicized middleware initiatives are the Open Software Foundation's Distributed Computing Environment (DCE) , Object Management Group's Common Object Request Broker Architecture (CORBA), and Microsoft's COM/DCOM (see Component Object Model (COM), DCOM, and Related Capabilities) [Eckerson 95].

## Technical Detail

As outlined in Figure 17, middleware services are sets of distributed software that exist between the application and the operating system and network services on a system node in the network.

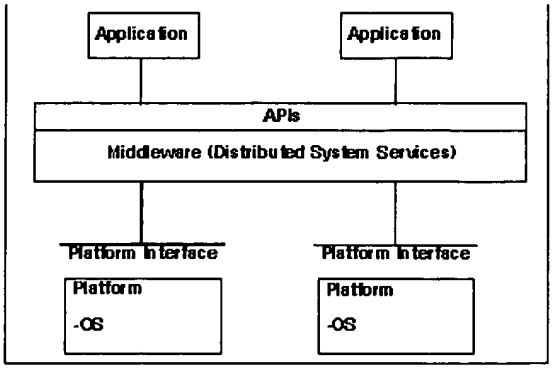


**Figure 17: Use of Middleware [Bernstein 96]**

Middleware services provide a more functional set of Application Programming Interfaces (API) than the operating system and network services to allow an application to

- locate transparently across the network, providing

interaction with another application or service
- be independent from network services
- be reliable and available
- scale up in capacity without losing function [Schreiber 95]

Middleware can take on the following different forms:

- Transaction processing (TP) monitors (see Transaction Processing Monitor Technology), which provide tools and an environment for developing and deploying distributed applications.
- Remote Procedure Call (RPCs), which enable the logic of an application to be distributed across the network. Program logic on remote systems can be executed as simply as calling a local routine.
- Message-Oriented Middleware (MOM), which provides program-to-program data exchange, enabling the creation of distributed applications. MOM is analogous to email in the sense it is asynchronous and requires the recipients of messages to interpret their meaning and to take appropriate action.
- Object Request Brokers (ORBs) , which enable the objects that comprise an application to be distributed and shared across heterogeneous networks.

## Usage Considerations

The main purpose of middleware services is to help solve many application connectivity and interoperability problems. However, middleware services are not a panacea:

- There is a gap between principles and practice. Many popular middleware services use proprietary implementations (making applications dependent on a single vendor's product).
- The sheer number of middleware services is a barrier to using them. To keep their computing environment manageably simple, developers have to select a small number of services that meet their needs for functionality and platform coverage.
- While middleware services raise the level of abstraction of programming distributed applications, they still leave the application developer with hard design choices. For example, the developer must still decide what functionality to put on the client and server sides of a distributed application [Bernstein 96].

The key to overcoming these three problems is to fully understand both the application problem and the value of middleware services that can enable the distributed application. To determine the types of middleware services required, the developer must identify the functions required, which fall into one of three classes:

1. Distributed system services, which include critical communications, program-to-program, and data management services. This type of service includes RPCs, MOMs and ORBs.
2. Application enabling services, which give applications

access to distributed services and the underlying network. This type of services includes transaction monitors (see Transaction Processing Monitor Technology) and database services such as Structured Query Language (SQL).
3. Middleware management services, which enable applications and system functions to be continuously monitored to ensure optimum performance of the distributed environment [Schreiber 95].

## Maturity

A significant number of middleware services and vendors exist. Middleware applications will continue to grow with the installation of more heterogeneous networks. An example of middleware in use is the Delta Airlines Cargo Handling System, which uses middleware technology to link over 40,000 terminals in 32 countries with UNIX services and IBM mainframes. By 1999, middleware sales are expected to exceed $6 billion [Client 95].

## Costs and Limitations

The costs of using middleware technology (i.e., license fees) in system development are entirely dependent on the required operating systems and the types of platforms. Middleware product implementations are unique to the vendor. This results in a dependence on the vendor for maintenance support and future enhancements. This reliance could have a negative effect on a system's flexibility and maintainability. However, when evaluated against the cost of developing a unique middleware solution, the system developer and maintainer may view the potential negative effect as acceptable.

## Index Categories

This technology is classified under the following categories. Select a category for a list of related topics.

| | |
|---|---|
| Name of technology | Middleware |
| Application category | Client/Server (AP.2.1.2.1)<br>Client/Server Communication (AP.2.2.1) |
| Quality measures category | Interoperability (QM.4.1) |
| Computing reviews category | Distributed Systems (C.2.4)<br>Network Architecture and Design (C.2.1)<br>Database Management Languages (D.3.2) |

## References and Information Sources

**[Bernstein 96]** Bernstein, Philip A. "Middleware: A Model for Distributed Services." *Communications of the ACM 39*, 2 (February 1996): 86-97.

[Client 95] "Middleware Can Mask the Complexity of your Distributed Environment." *Client/Server Economics Letter 2,* 6 (June 1995): 1-5.

[Eckerson 95] Eckerson, Wayne W. "Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications." *Open Information Systems 10,* 1 (January 1995): 3(20).

**[Schreiber 95]** Schreiber, Richard. "Middleware Demystified." *Datamation 41,* 6 (April 1, 1995): 41-45.

## Current Author/Maintainer

Mike Bray, Lockheed-Martin Ground Systems

## Modifications

25 June 97: modified/updated OLE/COM reference to COM/DCOM
10 Jan 97 (original)

---


URL: http://www.sei.cmu.edu/str/descriptions/middleware_body.html
Last Modified: 3 September 2003

# Remote Procedure Call

<*networking, programming*> (RPC) A protocol which allows a program running on one host to cause code to be executed on another host without the programmer needing to explicitly code for this. RPC is an easy and popular paradigm for implementing the client-server model of distributed computing. An RPC is initiated by the caller (client) sending request message to a remote system (the server) to execute a certain procedure using arguments supplied. A result message is returned to the caller. There are many variations and subtleties in various implementations, resulting in a variety of different (incompatible) RPC protocols.

Sun RPC is defined in RFC 1057 and ONC RPC in RFC 1831.

(2003-06-04)

Try this search on OneLook / Google

**Nearby terms:** remote login « Remote Method Invocation « Remote Operations Service Element « **Remote Procedure Call** » Remote Reference Layer » Remote Spooling Communication Subsystem » Remote Write Protocol

**Sun ==>**

# Sun Microsystems, Inc.

<*company*> One of the first, and now biggest, US computer manufacturers. They also manufacture in Europe. The Sun-2 and 3 series of workstations and servers were based on the Motorola 680x0 family of microprocessors and the Sun-4 series on the SPARC. Sun also produce their own version of Unix, originally called SunOS and now Solaris. Their Network File System has become the de facto standard for sharing files between Unix systems.

Quarterly sales $1403M, profits $78M (Aug 1994).

*Home*. *Sun World Online*.

Address: 2550 Garcia Ave., Mt. View, CA 94043 -1100 USA.

(1995-10-14)

Try this search on OneLook / Google

---

**Nearby terms:** Sun-3 Workstation « Sun-4 Workstation « sun lounge « **Sun Microsystems, Inc.** » SunOS » sunspots » sun-stools

---